一种应用于超算系统调度的作业并行参数优化方法*

张文帅[1] 李会民[1] 李京[2] 潘必才[3]

[1]（中国科学技术大学，网络信息中心，超级计算中心，合肥 230026）

[2]（中国科学技术大学，计算机科学与技术学院，合肥 230026）

[3]（中国科学技术大学，物理系，合肥 230026）

摘要： 随着高性能计算体系结构的发展，软件与硬件都具有了多层的并行结构。当不同纵向层级与横向分组的计算任务被划分到不同节点的不同处理器时，具有非常多的分配对应方式。用户越来越难以获取最佳的计算并行参数与硬件资源使用量。我们研究了一种优化方法，可以帮助用户自动化的确定最佳的应用并行参数与硬件使用量，使其可以高效率地扩展到大规模计算。此外，我们提出了一种将该优化方法与作业调度系统深度融合的方案，并获得了很好的应用效果。

关键词： 高性能计算　计算机集群　作业调度　运行时优化

分类号： TP338.6

A job parallel parameter optimization method applied to supercomputing cluster scheduling

Zhang Wenshuai[1] Li Huimin[1] Li Jing[2] Pan Bicai[3]

[1](Network Information Center, University of Science and Technology of China, Hefei 230026, China)

[2] (School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)

[2] (Department of Physics, University of Science and Technology of China, Hefei 230026, China)

Abstract: With the development of high-performance computing architectures, many software and hardware have a multi-layer parallel structure. A large amount of allocation schemes can be involved when users allocate multi-layered system resources to many computational tasks distributed in different vertical tiers and horizontal groupings. It is becoming increasingly difficult for users to determine the optimal parallel parameters and hardware resource usage. We investigate an optimization method which is helpful for users to automate the determination of the optimal application parallel parameters and hardware usage for high-efficient and/or large-scale computation. In addition, we propose a solution that deeply integrates the optimization method with the job scheduling system, which has produced excellent practical results.

Keywords: high performance computing computer cluster job scheduling runtime optimization

---

* 本文系中科院先导专项“地球大数据科学工程-云服务基础平台运行与安全项目”（项目编号：XDA19020102）的研究成果之一。

1 引言

以科学计算为目标，回顾计算硬件的结构发展，从 CPU 的同构多核处理器（SMP）构架到具有 NUMA 分区的异构方式；从单节点内的多路处理器胖节点到大规模的集群节点。可知，计算设备具有越来越多的并行层级，形成了越来越成熟的高性能计算设备。相应的，一般的科学计算程序，也会从简单的支持 OpenMP 并行发展到支持 MPI 并行，或者两者结合起来。特别是，近年来 GPU 异构设备及其上的应用开发也越来越完善。因此，可以观察到一个现象，软硬件在演进与成熟化的过程中变得越来越复杂，具有越来越多的并行层级与分组结构。

当不同纵向层级与横向分组的计算任务划分到不同分组的节点与处理器时，面临着巨大的可调空间。不同的分配方式可以导致明显不同的计算速度，也需要不同的计算资源。因为最佳的计算速度依赖于计算硬件也依赖于需要计算的目标问题，因此程序难以自动的预估处理这个分配模型，导致一些参数是必须通过试运行来完成速度评估。例如，在利用 FFTW 软件库执行 FFT 计算前，库函数需要开发者指定 FFT Plan，并可根据不同的硬件与数组大小选择不同的 FFT 计算方法与参数。虽然有一些研究工作致力于预测其中的最佳性能，并发展自动化的调优方法[1-2]，但在很多场景下，程序仍然使用 FFTW_MEASURE 甚至 FFTW_PATIENT 来实时的测试不同方法与参数的运行性能，使得程序在更长更大规模的运行算例下具有更好的效率。

在复杂的应用程序整体计算场景下，应用功能越丰富，其计算目标对象就越繁多，预估最优参数的方法就越发失效。特别是，即便解决了在程序内部自动优化并行参数的问题，也无法解决使用计算资源数量的配置问题，因为在计算开始前，这个计算资源数量必然是由用户或调度系统来提供的，而无法通过程序内部的参数来调整。因此，多数应用允许用户自己根据计算目标与硬件资源来调整设置这个任务划分方法与参数。这些参数一般在程序计算开始前在输入文件中给出，后面我们称之为应用并行参数，其仅影响计算任务的分组策略，原则上可以由用户或调度系统进行调整，且不对程序的计算结果造成影响。

以科学计算领域中使用最广泛的 VASP 应用为例，在我们过去的工作中本文中，尝试了通过输入参数结合历史运行数据来判断当前目标作业的计算时间，取得了一定的效果[3]，但是其精确度仍然有所不足，特别是 VASP 的总时间预测需要考虑到计算迭代的步数的影响，而不与单个迭代计算的速度直接相关，因此难以应用于运行速度与扩展性优化。本文中，通过设计一种针对不同硬件资源使用量与并行参数的优化方法，我们为用户提供了全面优化的运行参数。区别于常见的其他运行时优化工作，我们不仅为应用提供优化的输入并行参数数值，且优化应用的运行扩展性，提供最佳的并行规模，以及相应需要设置的资源申请数。

本文组织如下:首先在第一节中，结合 VASP 应用，介绍其并行逻辑、问题背景以及部分空间遍历测试。在第二节，我们给出本文的优化方法与优化指标。在第三节，本文结合几个测试实例，展示优化效果。第四节中，我们给出与作业调度系统结合的方法，并展示已有的部分结果。最后，给出总结。

2 VASP 应用并行模式与背景结果

本节中，我们给出 VASP 应用的基础并行模式，可以调节而不影响计算结果的并行参数，以及探索是否可以给出确定性经验模型的背景测试结果。

2.1 VASP 应用并行模式

VASP 是超算行业占用机时数最多的应用，对该软件的高效使用可以明显的提升超算集群的利用效率与用户工作产出。图 1 展示了 VASP 应用的任务并行划分的基本模式，表达了其三层并行下的任务与 CPU Core 的对应关系。可知，KPAR 表达了同时运行的 KPoint 任务个数，NPAR 表达了同时计算的 Band 任务个数，NCORE 表达了单个 Band 任务所使用的 CPU Core 核心数。因此，总的计算核心数为三个参数的乘积：KAPR x NPAR x NCORE，后续需要优化的并行参数也选择这三者，他们既决定了基本的并行任务划分方式，也决定了总的计算资源数。

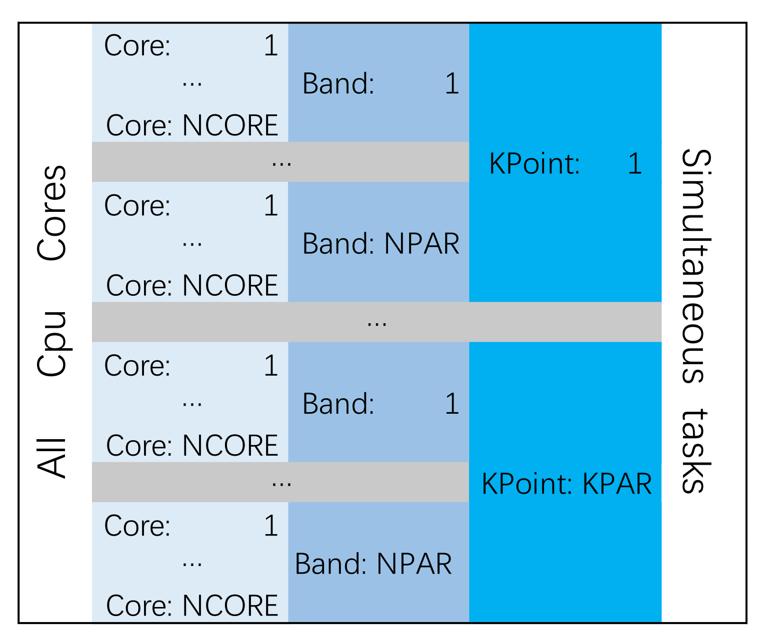


图 1 VASP 应用同时运行任务的并行划分模式

Figure 1 Parallel division pattern of tasks running simultaneously in a VASP calculation

2.2 背景测试结果

在介绍新的并行参数自动优化方法之前，我们首先介绍过去一些测试得到的经验结论[4]。在这里引用的文档[4]中，我们测试了 6 种硬件平台下的 5 种不同类型的计算算例。

测试结果表明，一些常用的设置参数，包括 VASP 官方文档中的设置建议，在一些计算场景下明显偏离最优值。例如，NPAR = 4 ~ approx. SQRT(number of cores) 在小体系多 K 点以及大尺度单 K 点算例下明显不适合；建议“不建议设置 KPAR > Computer Nodes”在小体系多 K 点下明显不合适等等。

此外，我们发现，在总计算核心数保持基本不变的前提下，NCORE 的最优取值空间相对 NPAR 的最优取值空间更加小，且单节点核心数应该可被 NCORE 整除为最优。结合图 1，这可以理解为，这个约束可以保证最小的并行计算资源组（NCORE 个 CPU Cores）可以在相同的节点内完成被分配的计算任务，因此可以大大减少节点间的通讯。测试表明将在一个 28 核心的节点上做计算时，将 NCORE 从 8 修改为 7，可以使得 MPI 通信耗时由 199 秒降为 131 秒，显著提升了并行计算效率。

如 University of Delaware 的 Geun Ho Gu 所言：“I have found that this instruction does not always hold up, and, really, this parameter is heavily dependent on the batch server / node configuration. So, it is wise to do your own test to optimize this parameter (and other parameters as well).”，其建议用户根据计算环境独立测试待计算算例的运行并行参数。以上测试除了使我们对 NCORE 我们有了一些明确的限制，缩小了最优空间，并不会为应用并行参数优化带来确定性的经验模型。

以上有关 NCORE 的经验特征也提示，针对具体应用，寻找其最小并行任务组，并根据整除规则（节点核心数）限制其取值空间，可以大大缩小需要优化的并行参数空间。我们基于此，在缩小的参数空间中进行了一个遍历测试，结果如图 2 中所示。

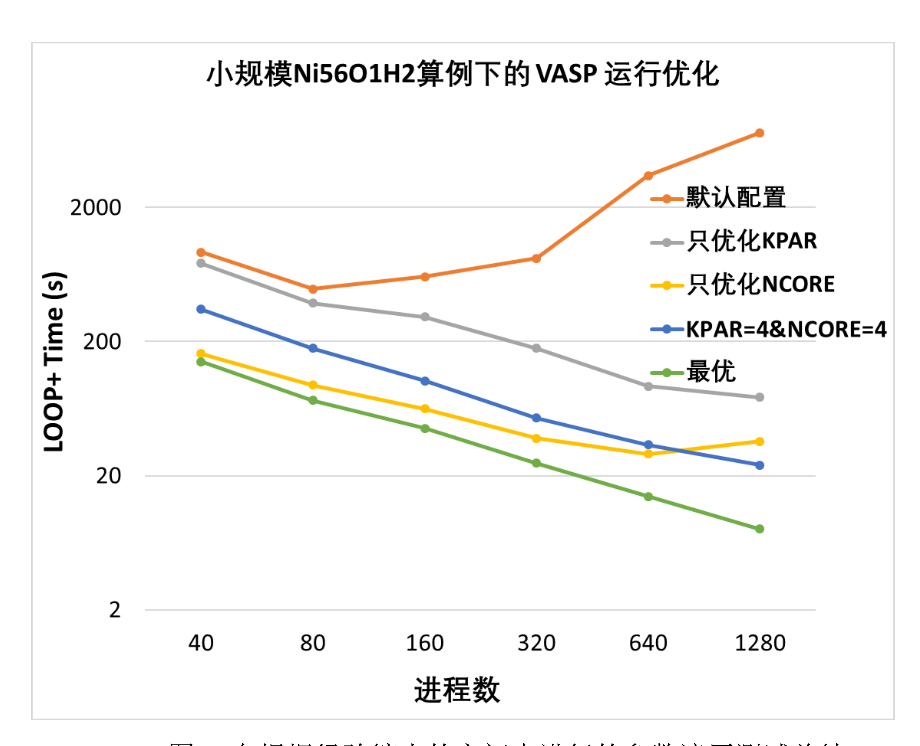


图 2 在根据经验缩小的空间中进行的参数遍历测试总结

Figure 2 Summary of parameter tests in an empirically reduced space

该算例具有 59 个原子，且只有 8 个约化的 K 点个数，属于一个较小的计算体系。因为遍历测试得到的数据结果很多，本文仅展示了其中的代表性数据。根据遍历测试结果，我们分别画出了 5 种配置或配置优化下的计算时间。

可以看到，默认配置下，计算时间很长，且在 100 进程数下具有最佳运行速度；分别只优化 NCORE 与 KPAR 时的速度有所提升，但最佳计算时间依然在 30-80 秒左右，且在 600 进程数下达到并行扩展瓶颈；在使用一些经验较丰富的用户所经常使用的经验值时，计算效果与前类似；当在全空间寻求最佳值时，我们看到，对于该小体系计算，VASP 可以比较高效的扩展到 1280 个核心，最佳计算时间小至 8 秒。最佳计算时间相比默认配置提升了约 1000 倍。

该遍历测试表明，某些计算算例的并行扩展效率测试显著依赖于是否进行过并行参数优化，获取一个具有复杂并行结构的计算的真实并行扩展能力，并不是一个简单的测试任务。特别需要小心操作的是，将具有不同并行参数优化程度的两个计算测试进行比较，并没有真实的意义。

3 优化方法

本文所述的优化方法中，最关键的是，提出了一种在两个不同运行配置使用不同数量的硬件资源的情况下判断两者优劣的量化指标。基于该指标，我们设计了一种兼顾效率与速度需求的优化步骤。

3.1 约化并行效率指标

我们将对比两个并行参数配置 Case1 与 Case2 的优劣的判定标准定为：

1）当两个不同的作业参数配置的运行成本相等时，取计算时间最短者为最优；

2）当两个不同的作业参数配置的运行成本不同时，基于约化并行效率指标 Er 值的高低来判断。

其中，约化并行效率$\boldsymbol{Er}$表示为：

$$E_r = E_p^{log_{R2/R1}^n} = (1/ECost)^{log_{ECost\cdot SpeedUp}^n}$$

$$E_p = \frac{R1\cdot T1}{R2\cdot T2} = \frac{1}{ECost}, SpeedUp = \frac{T1}{T2}$$

。

其中设定，该计算算例采用 Case1 时的运行资源数少于采用 Case2 时的运行资源数；R1 为 Case1 对应的运行资源数，R2 为 Case2 对应的运行资源数；T1 为 Case1 对应的作业计算时长；T2 为 Case2 对应的作业计算时长。所述运行资源数即为运行成本，一般可以以硬件成本来代替。

特别的，当 n 取 2 时，约化并行效率即为硬件成本（计算使用核心数）每增大为 2 倍时的相应并行效率。作为对比，并行效率指标难以在使用计算资源数不同的场景下，兼顾速度与资源利用效率的要求。例如，当比较两个扩展实例：使用 4 倍 CPU 核心且并行效率为 64%与使用 2 倍 CPU 核心且并行效率为 64%，显然具有不同的硬件资源利用效率，但却具有相同的并行效率值。约化并行效率 Er 公式中，不仅包含了 SpeedUp，还包括了 ECost，表明其不仅考虑了两个配置的速度差异，同时考虑了两者之间的资源成本使用差异，两者共同决定了两个参数配置的优劣。

3.2 优化步骤

为了公平的进行扩展效率对比，或者是为用户提供优化的计算速度与计算资源规模，我们需要简单的获取最佳应用并行参数的方法。

我们总结优化方法为如下步骤：

a）试运行一次 VASP 计算，在其完成初始化后即停止运行。

b）根据输出的约化 K 点数来确定 KPAR 参数的取值空间，其为约化 K 点数的因子。

c）根据输出的 NBANDS 数来确定 NPAR 的参数取值列表，后者同样为前者的因子。

d）根据单节点总核心数来确定 NCORE 参数的取值列表，后者为前者的因子或倍数。

e）根据算例规模，为 KPAR、NPAR、NCORE 猜测一个初始值。

f）以初始值为起点，按列表分别增加三个参数的取值，根据一个优劣标准来判定较优的并行参数，确定最佳的扩展路径。

g）当所有参数对应的效率指标不再高于预先设定的效率边界时，停止参数优化与并行扩展测试。

以上步骤中，最关键的是选择合适的优劣标准与效率指标，在本文中，我们介绍一种约化并行效率指标，可以更好的解决在使用不同硬件资源数的情况下判断两个并行参数设置的优劣。

4 计算并行参数与扩展性优化效果

本节中，分别展示三个计算算例下，针对计算并行参数与扩展性分别进行优化后的效果。以下结果基于 VASP6.3.3 版本，并且在 5.4 版本下也具有相同效果。

4.1 算例 1 优化结果

表 1 Ni56O1H2 体系下的运行参数优化结果

Table1 Optimization results for Ni56O1H2 calculation

Processes Number	Parameter: KPAR*NPAR*NCORE	*Er*	Loop + Time (s)
80	1 * 80 * 1	-	514
160	2 * 8 * 10	-	63
320	2 * 8 * 20	88%	36
320	4 * 8 * 10	98%	32
640	4 * 16 * 10	94%	17
640	4 * 8 * 20	84%	19
640	8 * 8* 10	107%	15
1280	8 * 16 * 10	94%	8

表 1 展示了 Ni5601H2 体系下的运行参数优化结果，表中第一行数据为用户的默认设置，后续几行为优化停止前的测试路径及相关结果。最后一列 Loop+ Time 表示预定的多个迭代计算的总时间。

该算例与本文第一节中的遍历测试算例相同，与前述遍历测试相比，本节使用的优化方法仅使用 7 次测试，每次测试只需要计算 5 个电子迭代步，即得到了最佳的运行并行参数，并将计算核心数扩展到 1280 核心，同时保持高达 94%的并行效率。此外，本计算算例具有离子迭代过程，因此 35 次电子迭代在全部计算任务中时非常小的一部分，本测试过程不会使得总花费机时变高。

4.2 算例 2 优化结果

表 2 Ni48C2O2 体系下的运行参数优化结果

Table2 Optimization results for Ni48C2O2 calculation

Processes Number	Parameter: KPAR*NPAR*NCORE	*Er*	Loop + Time (s)
200	1 * 4 * 10	-	906
200	4 * 2 * 5	-	515

400	4 * 2 * 10	143%	180
800	8 * 2 * 10	102%	88
800	4 * 4 * 10	85%	106
800	4 * 2 * 20	86%	105
1600	8 * 2 * 20	90%	49
1600	8 * 4 * 10	90%	49

表 2 展示了一种过渡态计算算例，其三个并行参数的乘积并不等于总的计算进程数，因为其会生成 5 个 image 同时进行计算，所以前两者数值之间相差 5 倍。表中第一行数据为用户的默认设置，后续几行为优化停止前的测试路径及相关结果。通过对该算例的并行参数优化，我们可以发现两个特征：1）在最终 1600 核心的优化结果下，NPAR 与 NCORE 分别取值 4 与 10，这与用户给出的 200 核心下的参数设置是相同的，但是 200 核心下的明显更优参数应该取值 2 与 5，体现了最佳并行参数会受到硬件资源使用量的较大影响，其数值在某个硬件资源量下取得最佳计算性能，在换一个硬件资源使用量下会明显的偏离最佳性能。2）在优化过程中可以常见到超线性加速，例如表中的 143%数值，这通常是因为用于比较的前一个计算配置不具有良好的并行状态。

本算例因为要计算多个过渡态，通常需要较多计算时间，本参数优化测试结果表明，用户可以通过使用更多的核心来加速计算，更快的取得研究进展，同时不增加计算机时费用。这也帮助超算集群实现更高的价值，避免其作为分割的小规模计算机来使用。

4.3 算例 3 优化结果

表 3 C100I48Cr16 体系下的运行参数优化结果

Table2 Optimization results for C100I48Cr16calculation

Processes Number	Parameter: KPAR*NPAR*NCORE	*Er*	Loop + Time (s)
80	1 * 80 * 1	-	1087
160	1 * 16 * 10	-	127
320	1 * 16 * 20	113%	56
1600	5 * 16 * 20	97%	12
800	1 * 40 * 20	73%	34
640	1 * 16 * 40	82%	34
1600	5 * 16 * 20	97%	12

表 3 展示了一种较多原子数较少 K 点个数（5 个约化后 K 点）的计算算例，表中第一行数据为用户的默认设置，后续几行为优化停止前的测试路径及相关结果。因为 K 点数较少，导致 KPAR 参数空间较小，因此仅用了 5 次测试，即获得了优化结果，本次优化的效果显著，为用户原始计算提供了接近百倍的加速。因为本算例也是具备离子迭代过程，作业总时间会较长，因此这个加速配置具有非常好的应用机会。

5 与作业调度系统的结合方法与应用效果

本节中，我们提出一种将前述并行参数优化方法集成到作业调度系统中的流

程，如图 3 所示，用户只需要提交作业并等待优化结果即可。

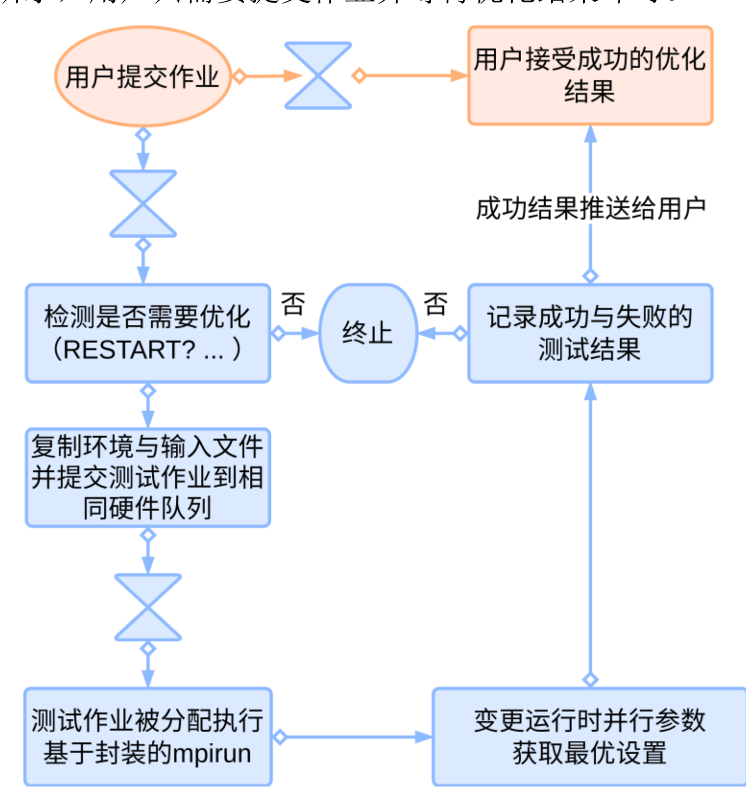


图 3 作业调度系统与并行参数优化方法集成流程

Figure 3 Combining job scheduling system with the parameter optimization module

实践中，图 3 中用于执行优化的测试作业可被赋予可被抢断属性，并立刻在集群中当前闲置的节点中运行，因为优化测试作业的运行时间通常比较短暂，其有较大概率在可能出现的被抢断事件前完成执行。因此该类作业可以在不影响用户其他作业的前提下工作，并较少集群节点的闲置时间，增加集群的利用效率。此外用户接受成功的优化结果后，可以选择自动的采用优化的并行参数，也可以选择被通知并手动采纳。特别是，可以通过为优化过程增大扩展性测试的规模限制，可以使用户更多的接收到大规模计算的设置建议，并通过自动更新作业规模，来动态的调整小的用户作业为大规模计算作业，充分实现大规模超算集群设备的建设初衷。

我们在中国科大超算中心集群中实施了图 3 中所示流程，限于集群规模，并未进行大规模的扩展性优化，仅允许在用户原始计算资源申请量的 1.5 倍范围内进行优化测试。测试的结果如图 4 中所示，我们选择连续的 1400 个作业的优化结果，每 100 个作业为一组，展示优化后的计算加速效果与平均机时消耗。可以看到，调度系统的自动参数优化插件可使 VASP 作业平均加速约 1.6 ~ 3.2 倍，同时使得 VASP 作业的平均机时消耗减少 15% ~ 35%之间。

更多的测试结果仍有待统计研究，我们希望可以更进一步容许用户不在选择队列与资源申请数额，通过作业调度系统来为用户作业自动配置最佳的资源申请数额，这在 Web Portal 模式的超算使用方法得到广泛采用后，可以得到很好的实施。特别是，因为采用约化并行效率指标后，我们可以通过调节该数值边界，控制计算速度与资源利用效率的平衡点位置。当将约化并行效率 Er 的容许边界调整为较高值时，即为计算作业提供“效率优先、速度次之”的选项；将 Er 的

容许边界调整为较低值时，即为计算作业提供“速度优先、效率次之”的选项；将 Er 的容许边界设置为中间值时，即实现了一种默认的均衡选项。

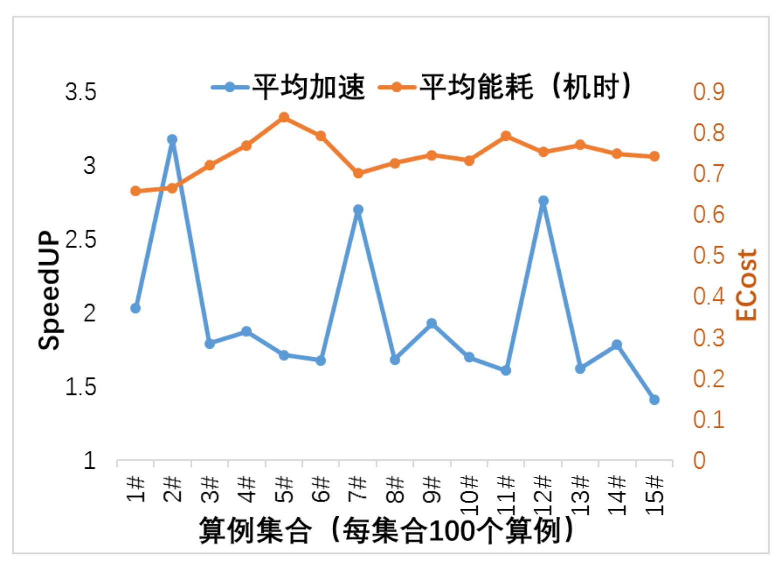


图 4 全自动作业优化效果

Figure 4 Results of fully automated parameter optimization

6 总结

前文描述了在多层结构的软硬件计算场景下，通过试运行来自动化的帮助用户获取最佳运行速度与最佳扩展度的必要性与可行性。特别发现，计算算例的并行扩展效率测试的好坏，显著依赖于并行参数的优化程度。针对此目的，本文提出了优化方法与判定标准，提出了一种约化并行效率的指标，可以兼顾计算速度与计算效率的要求，能够很好的帮助多数 VASP 作业在同等资源下显著提升速度的同时，在保证计算效率的情况下大大提高了计算的扩展度与极限速度。通过将该优化方法跟作业调度系统结合，我们展示了该解决方案的应用潜力。

本文提出的优化方法可以帮助用户使用更多的核心来加速计算，更快的取得研究进展，同时不增加计算机时费用。这更帮助超算集群实现更高的自身价值，避免其被分割为小规模计算机来使用，这也本是大规模超算集群建设的初衷。

此外。希望本方法可以为应用的 Web Portal 化提供重要的一环，使用户无需关心并行参数与计算资源数，即可在计算速度优先与资源利用效率优先之间的任一平衡位置进行配置计算。

参考文献：

[1] Gu L, Li X. DFT performance prediction in FFTW[C]//International Workshop on Languages and Compilers for Parallel Computing. Springer, Berlin, Heidelberg, 2009: 140-156.

[2] Guarrasi M, Erbacci G, Emerson A. Auto-tuning of the fftw library for massively parallel supercomputers[J]. PRACE: Partnership Advanced Computing Europe, Tech. Rep, 2014: 1-12.

[3] Wu Gui-bao, Shen Yu, Zhang Wen-shuai, et al. Runtime Prediction of Jobs for Backfilling Optimization. Journal of Chinese Computer Systems, 2019, 40(1), 6-12.

[4] http://scc.ustc.edu.cn/_upload/article/files/cd/05/bfefd0ec4fc08d2843fded7 7bca5/3118fa63-8f2d-4fd5-8582-161dfe1342e8.pdf

(通讯作者：张文帅 E-mail:wszhang@ustc.edu.cn)